DEVELOPERS AVS INSTALLATION/ RELEASE NOTES

Release 5.5 (50.86 / 50.88)
November, 1999

Advanced Visual Systems Inc.

Part Number: 330-0150-02 Rev L

Advanced Visual Systems Inc.
300 Fifth Ave.
Waltham, MA 02451

Printed in U.S.A.

TABLE OF CONTENTS

1 Developers AVS Overview

Introduction 1-1
Developing Applications 1-1
Deploying Applications 1-2
Product Licensing 1-2
Product Software Requirements 1-5
Problems fixed in AVS 5.5 1-5

2 Installation

Installing Developers AVS 2-1
License Requirements 2-2
Distribution Contents 2-3

3 Developing Applications

Introduction 3-1
Building the Examples 3-1
Example Applications 3-1
- app1: Basic Application Program 3-2
- app2: Sending Command Line Arguments and CLI Commands to AVS 3-3
- app3: Motif-Based Interface 3-4
- app4, app5, and app6: Motif-Based Interface with Reparented Drawing Area 3-5

Common Application Features 3-7
- Suppressing the AVS Menu 3-7
- Licensed Applications: Defining a FLEXlm Licensing FEATURE String 3-8
- Unlicensed Applications (OEM-Supplied Licensing) 3-9

Hook for License Validation 3-9
Unlicensed AVS: Enabling the Network Editor 3-9
Application Makefiles 3-10
Deploying an Application 3-12
Optimizing Geometry Modules: Linking Modules with the Kernel 3-15
Procedure Summary 3-16
Changes to the Module's Source Code 3-17
Changes to the Application Main Routine 3-18
Changes to the Module Library File 3-18
Changes to the Network File 3-19
Changes to Makefiles 3-19
Special Case: Duplicate Module Names 3-19

4 Module Source

Two Module Source Products 4-1
Compiling Modules 4-1
Module to Module Source Mapping 4-1
Module to Module Binary Mapping 4-1

CHAPTER 1

DEVELOPERS AVS OVERVIEW

Introduction

This document describes the Developers version of Advanced Visual Systems Inc.'s AVS 5.5 product as it is implemented on all platforms.

The Developers AVS product is designed to provide application developers with the ability to develop and deploy AVS-based applications to their users. Although these applications will use various AVS capabilities, end-users will often be unaware that the application they are using is built on top of AVS.

AVS 5.5 Notes: As a convenience to the reader, changes made for AVS 5.5 are summarized at the beginning of each chapter in "AVS 5.5 Notes" blocks.

This release now includes the Linux Developer's AVS product which is unlicensed (you will need to buy the standard AVS release for Linux separately); several Developer's AVS bugs were fixed; the AVS/Animator is now unlicensed and source is provided in the standard release; some new licensing options exist for greater convenience (contact AVS Customer Support for details). For more information, see the Standard AVS release notes document for your platform.

Developing Applications

During the development phase, the developer uses the standard version of AVS to create modules, build networks, and generate an application.

The components of an application built with AVS include:

- AVS modules
- AVS networks
- AVS runtime support files
- Developer-specific application code

The developer-specific application code can take one of several forms. In all cases, the developer application consists of one or more AVS networks, and one or more AVS modules. The most common form of application uses a pull-down menu interface, with menu button selections tied to AVS network creation and modification. The developer can also build an application with AVS as a network of modules whose user interface has been re-configured with the AVS Layout Editor. Alternatively, the developer can construct an application that drives AVS using the Command Language Interpreter (CLI), allowing the use of developer-supplied or third-party user interface toolkits.

Deploying Applications

Once the application is developed with the standard version of AVS, it must be packaged and deployed by the developer. The deployed application is based on a version of the AVS executable created by the customer. This program performs its own initialization and then makes a procedure call into the AVS Runtime Library (*libavs.a*) to initiate AVS.

An application packaged in this way allows AVS modules to be executed and networks to be built. It has access to all of the AVS Viewers.

In the RuntimeA version, two AVS components will not be accessible to a person using the application:

- the AVS Network Editor, and
- interactive Command Line Interface (CLI) commands. (The application may still issue CLI command strings with the **AVScommand** function call.)

In the RuntimeB version, the Network Editor will be accessible to a person using the application. The interactive CLI is not.

Selection of RuntimeA or RuntimeB versions is controlled by licensing.

Product Licensing

AVS 5.5 Note: The Linux platform is NOT licensed at all so the discussion below concerning licensing is not applicable. However, if you wish to use the Linux platform as a Developer's AVS customer you need to buy the Standard AVS product for Linux separately through your AVS sales representative.

In order to protect against uncontrolled distribution of the AVS product, a licensing protection scheme is required. This "licensing" involves three cases:

Licensing AVS for You, the Developer

You develop your application using the standard AVS distribution. You obtain and install the license that enables you to run AVS as described in the *UNIX Installation and Release Notes* book that accompanies the standard AVS distribution media. Standard AVS is licensed using FLEXlm .

The FEATURE line that enables AVS in the *license.dat* file that you obtain from Advanced Visual Systems has the FEATURE name "AVS".

Licensing Your Internal Runtime Applications

When you purchased Developers AVS, you obtained at least one **runtime** license.

When you make the example applications, *app1*, *app2*, etc., or when you create test versions of your own applications, you link them against the AVS library *libavs.a*. When linked against *libavs.a*, the application will automatically include FLEXlm.

An application linked against *libavs.a* requires a runtime license to run. This runtime license is represented by a FEATURE line in the *license.dat* file that you obtain from Advanced Visual Systems Customer Support.

The default FEATURE name for RuntimeA (no Network Editor) is "AVS-Runtime".

RuntimeB (with Network Editor) developers must define a FEATURE name that contains the suffix string "-NE" and register it with Advanced Visual Systems Customer Support to enable the Network Editor. The section "Defining a FLEXlm Licensing Feature String" in the "Developing Applications" chapter explains how to do this.

Licensing Your Customers

The applications that you develop and provide to your customers require an AVS **runtime** license. This runtime license will also be a FLEXlm system license. The general procedure is:

1. When you create the application, you must specify a FLEXlm FEATURE string that uniquely identifies your application in the application program. You are renaming the application from the default "AVS-Runtime" to your own name. The section "Defining a FLEXlm Licensing FEATURE String" in the "Developing Applications" chapter describes how to do this.
2. You must register this FEATURE string with Advanced Visual Systems Customer Support. Contact Customer Support through e-mail at support@avs.com or through your local Sales office.
3. When you link your application to the AVS binary *libavs.a*, it will automatically include FLEXlm.
4. When you package your application, you must include the FLEXlm licensing utility binaries (*lmgrd* license manager daemon, *avs_lmd* vendor daemon, *lmstat* status command, etc.) on your distribution media. These FLEXlm utilities are provided on the Developers AVS CD in the directory *license* for redistribution with your application. You *must* use the Developers AVS versions of these utilities, not the versions on the regular AVS release media.

   Your customers will use these FLEXlm utilities to control access to your application.
5. Also with your application, you include instructions that your customers will follow to install the FLEXlm licensing mechanism and obtain a valid *license.dat* file.

The *nroff/troff* source and PostScript files for Advanced Visual Systems own licensing instructions (chapters 4 and 5 of theAVS5.02 AVS *Installation and Release Notes*) are included with the Developers AVS release. You can adapt, edit, or rewrite this material as you require for your customers. NOTE: These instructions were updated and reorganized for AVS5.3 and beyond - please contact support to request the newer versions if you are interested.

The *nroff/troff* source is in the Developers AVS directory hierarchy under *license/flexadmin.me* and *license/flexuser.me*. Note that these files contain Advanced Visual Systems-specific *nroff/troff* macros that will not interpret on your system. You will need to edit them to conform to your own local formatting codes and standards. If necessary, Advanced Visual Systems will provide its *nroff/troff* macros upon request.

The corresponding PostScript to these files is in the Developers AVS directory hierarchy under *license/admin.ps* and *license/user.ps*.

6. When your customers receive the application, they go through essentially the same process that you went through to install AVS licensing. Before they will be able to run your application they must:

   1. Obtain a FLEXlm-format *license.dat* file.

   • To obtain the *license.dat* file, they apply to you for the licenses, including all of the necessary machineid information, as described in the licensing documentation that you provided to them.

   • You, in turn, relay the request to Advanced Visual Systems Inc. Customer Support using a "Runtime End-User License Request" form.

   • Fill out the "Runtime End-User License Request" form, then e-mail it or send it by facsimile to Advanced Visual Systems Inc. at the address or number on the form. Prompt turnaround is provided.

   • When you receive the license codes in the *license.dat* file, you forward it, with any additional instructions, to your customers.

   2. Install the FLEXlm licensing mechanism using the FLEXlm utility binary files you provided in Step 4 above.

Alternative Licensing Mechanisms:

Some developer customers would prefer to use their own licensing software in place of FLEXlm and avoid contacting Advanced Visual Systems to generate end user licenses. This option is available through special agreement with Advanced Visual Systems and the use of replacement binary libraries. Please contact your sales representative for more information. **There are also now some new variants on the FlexLM licensing scheme for licensed users; contact AVS Customer support for more information.**

Product Software Requirements

The Developers AVS product requires certain graphics products in order to link the example applications as well as user applications. Depending on the platforms, you may need to purchase the appropriate graphics libraries (XGL, PHIGS, OpenGL, etc) from the product vendor. Stub libraries are provided in some cases in the *avs_developer/lib* directory.

Problems fixed in AVS 5.5:

- **CFS 7793/7812 "Developer's AVS: AVS-Runtime encryption problems"**

AVS 5.4 and earlier releases used a different encryption code for runtime licensing than what was used in the standard licenses. With the upgrade to FlexLM 4.x in the AVS 5.3 upgrade, support for this second encryption code was lost causing problems with existing licenses in the field and temporary difficulty making working licenses. This has been fixed for AVS 5.5 and offers the Developer customer two options:

- AVS 5.4 and earlier: AVS-Runtime features use the alternate encryption code and must be run using the old FlexLM (2.4c) AVS *lmgrd* daemons provided with AVS 5.02. AVS 5.3 and AVS 5.4 use these codes but their *lmgrd* daemons won't recognize them.
- AVS 5.5 and AVS 5.4 patch: AVS-Runtimes are generated using AVS 5.5 or AVS 5.4 with a patch available from AVS Customer Support. They use the same encryption codes as Standard AVS, and must run using FlexLM 4.x or higher daemons. AVS 5.5 and beyond will make this the default code. Existing AVS 5.4 or earlier runtime licenses should be reissued by AVS Customer Support.

If you are affected by this issue, please contact AVS Customer Support (support@avs.com) for more information or new licenses.

- **CFS 17869 "Developer's AVS - *liboglx.a* missing"**

AVS 5.4 expanded support for OpenGL based rendering and with AVS 5.5 this is now the default for all platforms. An important library needed by Developer's AVS customers was not included in the AVS 5.4 release, *liboglx.a*. This has been fixed for AVS 5.5 on all platforms.

- **Problem "Developer's AVS - Makelib files incorrect"**

Several platforms had slightly incorrect settings in these files. These issues have been resolved.

Problems fixed in AVS 5.5:

CHAPTER 2

INSTALLATION

Installing Developers AVS

AVS 5.5 Notes: The Linux platform is now provided for Developer's AVS 5.5. However, if you wish to use the Linux platform as a Developer's AVS customer you need to buy the Standard AVS product for Linux separately through your AVS sales representative.

The Developers AVS product consists of two components:

Standard AVS
The first component contains a copy of the standard AVS 5.5 release. This is a multi-platform CD from which you will extract the version of AVS for the target platform. It also includes a copy of the "Cool CD", which provides a snapshot of the content of the International AVS Centre's AVS module archive.

Developers AVS
The second component contains Developers AVS files for a variety of platforms. There are two different versions of the CD: either the standard licensed version of Developers AVS, or the unlicensed version. (Note: If you require the unlicensed version, please contact your sales representative).

You must install both components.

To Install Standard AVS

Install the standard AVS 5.5 release according to the instructions in *AVS on UNIX Workstations: Installation/Release Notes.* Linux users should read the release notes document provided with their release.

AVS 5.5 Note: The Linux platform is NOT licensed at all so any discussion below concerning licensing is not applicable.

When you contact Customer Support for licenses as described in the documentation, be sure to mention that you need both standard AVS licenses and AVS-Runtime licenses. Verify that the license.dat file that you receive has FEATURE lines for both AVS and AVS-Runtime (no Network Editor).

If you have the RuntimeB option (Network Editor enabled in applications), then you must give Customer Support a FEATURE name that includes the suffix "-NE" and define it in your application as described in "Defining a FLEXlm Licensing FEATURE String" in the "Developing Applications" chapter before the Network Editor will be enabled in your application.

To Install Developers AVS

Use the standard instructions provided for installing standard AVS using the Developers AVS CD.

1. Mount the CD, select a destination directory, run the *install.avs* script to select and install the desired version(s) of Developers AVS.
2. To verify the installation, you should build the example applications. See the "Developing Applications" chapter for instructions.
3. In order to run the example applications or an application that you build yourself, you must have obtained an "AVS-Runtime" license from Advanced Visual Systems Inc. Customer Support as noted at the beginning of this section.

AVS 5.5 Note: Linux is provided as a standard product archive that is installed using *install.avs*. The RPM format is not used for Developer's AVS but is supported for Standard AVS.

To Install Module Source

You install the Module Source in exactly the same way as you install the rest of Developers AVS. For historical reasons there are two versions of the module source:

- AVS_MOD_SRC9 - 64 bit platforms (Compaq Tru64 UNIX, SGI 64 bit)
- AVS_MOD_SRC8 - all other platforms.

Follow the instructions in the previous section, but specify AVS_MOD_SRC8 or AVS_MOD_SRC9 when presented with the menu of products. See Chapter 4, "Module Source" for more information.

License Requirements

AVS and the applications that you build with Developers AVS are licensed using the FLEXlm Flexible License Manager system (UNIX platforms only, not including Linux). You obtain the licenses from AVS Customer Support or your local AVS distributor. AVS and any applications that you build will not run until you have obtained a license. You will need these licenses before you can run AVS or your applications:

Standard AVS License

You develop applications using the standard AVS distribution. Follow the licensing instructions in *AVS on UNIX Workstations: Installation and Release Notes* to obtain and install this license. Its FEATURE string is "AVS".

AVS-Runtime License

The applications that you build also require a FLEXlm license. The default FEATURE string for the RuntimeA (no Network Editor) version is "AVS-Runtime" by default; you may select an alternate FEATURE string and register it with Customer Support to make unique licenses that will only work with your application.

There is no default FEATURE string for the RuntimeB (Network Editor) version. You should tell Customer Support what your FEATURE string will be. It must include the suffix "-NE".

You should obtain these runtime licenses at the same time you get the standard AVS license. You install them in the same way as the standard AVS license.

Note: Those few developers who have special arrangements with Advanced Visual Systems for unlicensed Developers AVS do not need these runtime licenses.

Distribution Contents

All files for this distribution are kept in the directory in which you installed Developers AVS.

The following is a detailed listing of the contents of this release.

Installation Script

install.avs

Script to select product to install - Developers AVS or module source.

avs_developer

The Developers AVS directory.

avs_developer/examples

Several example applications along with a sample Makefile.

avs_developer/include

Makelib.$(PORT) files that provide platform-specific linking information.

avs_developer/lib

The developer libraries needed to link applications.

avs_developer/license

FLEXlm licensing binaries and documentation (licensed versions only).

avs_developer/tools
Installation tools to facilitate deploying an application.

Module Source

module_source
The AVS Module Source.

module_source/Makefile
Makefile for making the entire module source, or just links for making individual modules.

module_source/README.mod_src
ASCII text file that explains how to make the modules.

module_source/avs_library
Target for making the entire module source.

module_source/kernel
Include files not in the standard AVS necessary to make the modules.

module_source/lib
Library files not in the standard AVS necessary to make the modules.

module_source/modules
Source code to the supported AVS modules.

module_source/mod_map
Contains a table that shows which modules are in which module source files.

module_source/mod_map2
Contains a table that shows which modules are in which module files.

module_source/opt
Source and include files not in the standard AVS distribution necessary to make some modules.

module_source/rflib
Various files necessary to make the **read field** module.

module_source/unsupp_mods
Source code to some of the unsupported AVS modules.

Source Code

Source code provided with this release consists of example applications that illustrate several ways in which to create an AVS application and the Module Source. These are described in detail in the next chapte

CHAPTER 3

DEVELOPING APPLICATIONS

Introduction

Once you have installed the Developers AVS CD, you should build the example applications. These serve both to verify the installation, and to provide a working example of how to build and deploy an application. This chapter describes how to build the examples and make your own application.

AVS 5.5 Notes: No changes have been made in this chapter. As noted elsewhere any licensing issues do not apply to the Linux platform which is unlicensed.

Building the Examples

The examples are found in the *examples* subdirectory of the install area. They can be made in place.

You need to define where the developer area is using an environment variable, **DEV_DIR**. For example, if the developer area was in */users/me/myarea* then the following (C shell) should build the examples:

```
setenv DEV_DIR /users/me/myarea/avs_developer
make -e
```

By default, the examples assume that AVS is installed in */usr/avs*, or that */usr/avs* is a link to the directory in which AVS is installed. If this is not the case, then the **AVS_PATH** environment variable should be set to specify where AVS is. For example, if AVS is installed in */users/me/avs* then the examples would be built as follows using the C shell (*csh)*:

```
setenv AVS_PATH /users/me/avs
make -e
```

Example Applications

There are six examples provided in this release. Each shows a different approach to building an application on top of AVS.

app1

This application is a simple example of starting AVS by running a CLI script to start an AVS network using standard AVS modules, and then enter the Geometry Viewer.

app2

This is a more sophisticated example. It shows how command line options can be processed to add support referencing where the application is installed, and how to embed a series of CLI startup commands within the application without the need for an external script.

In this example, the CLI commands are used to set an application path variable and then load a driver module *app2_mod*. This custom built coroutine module creates pulldown menus that select other scripts to run. These simple scripts present different module control panels or exit from the application. This application is also featured in an example of how to deploy an application.

app3

This extended example is similar to *app2* but presents a Motif-based user interface to AVS. It will only build on systems that provide Motif libraries. *app3* is done in the style of existing AVS applications—there is a control panel and a separate, free-floating Geometry Viewer window.

app4
app5
app6

These three examples are similar to one another. Like *app3*, they show a Motif-based user interface. However, where *app3* places the controls and Geometry Viewer window in separate windows, *app4*, *app5*, and *app6* creates controls and a viewer area in a single panel. This user interface style is closer to that of most Motif applications.

app4's drawing area is a reparented Geometry Viewer window. *app5* shows a reparented Graph Viewer window. *app6* shows a reparented Image Viewer window. They will only build on systems that provide Motif libraries.

Once the examples have been built, you should run them to see how they work. Start each by giving its name with no arguments.

app1: Basic Application Program

As can be seen from examining one of the application examples, the basic application is just a small *main* function that calls the *avs* function to initiate AVS. For example, this is the contents of the *app1.c* example:

```
main()
{
   int argc;
   char *argv[3];
   extern char *strdup();
   argc = 3;
   argv[0] = strdup("avs");
   argv[1] = strdup("-cli");
   argv[2] = strdup("script -play app1.scr");
   avs(argc, argv);
}
```

The program creates a number of command line arguments and then feeds these into the *avs* function as if it were a direct call by the user. This allows the application to start AVS and present a particular network or module. All command line options are recognized with the exception that the **-cli** option will not enable the interactive CLI interpreter; and the Network Editor is not available. (The "Suppressing the AVS Menu" section below describes how to prevent the main AVS menu from appearing, even transiently.)

app2: Sending Command Line Arguments and CLI Commands to AVS

The application's main function can also accept command line options from the user and pass them on to *avs()*, either filtering out options or adding some of its own. This would permit the end user some additional control over the behavior of AVS. This is illustrated in the *app2* application:

```
#include <string.h>

main(argc,argv)
int argc;
char **argv;
{
   int i;
   char temp[512], default_path[128];
   int local_argc = 1;
   char *local_argv[5];

   extern char *CLIinitial_commands;
   extern int CLIinitial_output;

   /**********************************************************************/
   /* Assume that if -path is given that the application has been installed.*/

   local_argc = 1;
   local_argv[0] = strdup("app2");
   strcpy(default_path,".");

   for (i = 0; i < argc; i++)
      if (!strcmp(argv[i],"-app_path")) {
      local_argv[ local_argc++ ] = strdup("-path");
      local_argv[ local_argc++ ] = strdup(argv[++i]);    /* pathname */
      strcpy(default_path, argv[i] );
      }

   /*********************************************************************/
```

```
    /* Set up initial conditions
       1) Create a CLI variable called AppPath that other scripts and networks
          can reference. Default is the current directory but an environment
          variable named APP_PATH will override this value.
       2) Make an instance of the control module.

       Store these commands into CLIinitial_commands to be executed during
       initialization.
    */

    sprintf(temp,"var_set AppPath -env APP_PATH %s \n\
    module \"App2_Mod\" -alias app2 -tag app2 -xy 268,102 -ex $AppPath/app2_mod",
    default_path);

    CLIinitial_commands = temp;

    /* Control normal output messages when executing CLIinitial_commands */
    /* 0 if no output desired, 1 for stdout */
    CLIinitial_output = 0;

    /*********************************************************************/
    /* Pass on startup arguments and startup avs */
    avs(local_argc, local_argv);
}
```

Note how, in this example, the application prepares a block of CLI commands in a string buffer and then stores this buffer into an AVS global variable named *CLIinitial_commands*. This avoids the need for a startup script and allows for application-specific CLI variables to be preset for later reference. It also illustrates one way in which the application can learn where its "home" directory is, handling both the situation where it is relying on *$AVS_PATH* and when it is has been "deployed" with its own local subset of AVS components.

app3: Motif-Based Interface

The AVS interface mechanism uses its own widget set. This widget set is defined largely in the *liblui* library.

Current windowing applications will typically want to use a Motif-based user interface. *app3* demonstrates how to create an application that has a Motif-based interface. The interface has three parts:

- A control menu that reads files, exits the application, and selects the viewing techniques (downsize, orthogonal slice, field to mesh, etc.)
- A Geometry Viewer output window.
- New control panels that appear for each menu bar choice selected. For example, if you select "File", a file browser panel appears. Selecting "downsize" causes a downsizing panel to appear.

The control panels, pulldown menus, and module control widgets (sliders, etc.) are all Motif widgets.

Several files make up *app3*:

app3.c
: The application main routine that calls AVS.

app3.scr
: The script file that reads in the application module, app3_mod.

app3.net
: The network file containing AVS modules that make up the saved application.

app3_mod.c
: The app3_mod module's main routine. Besides containing the module description function, it opens the display, initializes the toolkit, establishes the main control panels, reads in the network file, and contains the main event loop.

app3_menu.c
: Contains the X Toolkit and Motif calls that create the main control menu. It also associates the menu's selections with callback routines that create pop-up controls.

app3_popups.c
: Contains the callback routines that create the popup panels. It also associates the controls on each panel with a callback routine that will be invoked if the widget (slider, toggle, etc.) that represents a module parameter is changed.

app3_callb.c
: Contains the "parameter has changed" callback routines. These routines send CLI commands to the actual AVS modules.

app4, app5, and app6: Motif-Based Interface with Reparented Drawing Area

app4, *app5*, and *app6* demonstrate how to create a single Motif interface panel that contains the menu bar, a drawing area reparented from an AVS viewing window (*app4*: Geometry Viewer; *app5*: Graph Viewer; *app6*: Image Viewer), and all of the Motif widgets for its associated module control widgets. (*app3*'s menu bar and its module control popups are all on different Motif panels and the viewing area is not a Motif widget at all.)

The file structure of these applications' components are similar to *app3*'s. The main functional difference is reparenting the viewer window.

The following procedure is used to reparent an AVS viewer window to a Motif drawing area.

1. Create the network file that is the application.
2. Edit the application network file. Find the line that says:

   panel "geometry viewer.user ... OR
   panel "graph viewer.user ... OR
   panel "image viewer.user ....

   Just prior to that line, insert a line that says:

   panel app_win -w panel -p ui -hide -xy 780,10 -wh 500,500

   This creates a hidden parent window that will be used in the application to find the X window of the appropriate viewer. The "-wh" specifies the size of the application viewer window. This size should be consistent with the size of the Motif drawing area created in the application.

   Edit the original panel line for the Geometry Viewer or Graph Viewer, adding "-p app_win". This will make the viewer's parent the hidden panel that was just created. If this isn't done, you will not be able to find the viewer window in the application.
3. Use the utility **getXWindow** in the application to find the X window of the viewer. The input to the routine is the name of the hidden panel created in the network file (e.g., *app_win*). The output of this routine is the window to reparent to the Motif drawing area.

   The operation of this routine is not the same for all of the viewers. For the Image Viewer, the **getXWindow** utility returns the child of the hidden panel window. For the Graph Viewer or Geometry Viewer, the **getXWindow** utility descends the window hierarchy until the bottom is reached. This window is returned. Consult the appropriate source code example for the code of the **getXWindow** routine.

   In either case, the window returned by the utility is the window to reparent to the Motif drawing area.
4. After creating the Motif drawing area in the application, add resize and expose callbacks. These callbacks will reparent the window returned from **getXWindow** to the Motif drawing area the first time they are called. These callbacks can go on to do other application specific work but must perform the job of reparenting for the AVS viewer window to show up in a Motif drawing area.

General Notes

The routine **getXWindow** is found in each of the application modules (i.e., *app4_mod.c*, *app5_mod.c* and *app6_mod.6*). As mentioned previously, this routine is different for the Image Viewer.

The resize and expose callbacks to be added to the Motif drawing area are found in each of the application modules under the names **resizeIt** and **exposeIt.**

The creation of the Motif drawing area for each application is found in the file *app*_menu.c* in the routine **build_app*()**.

Switching Between Hardware and Software Renderers

As coded, *app4* does not support switching between the hardware and software renderers. Such a switch destroys the viewer window that has been reparented to the drawing area. It is possible to support hardware/software renderer switching. The general approach is:

- To see if the renderer window has been destroyed, put a conditional into the **exposeIt** callback routine that checks to see if the number of children of the drawing area is 0 (reparented window has been destroyed).
- Then, find the window id of the new viewer window using **getXWindow**, and reparent it to the drawing area as before.

Common Application Features

This section describes how to perform common functions within applications.

Suppressing the AVS Menu

In the example code as provided, the main AVS menu will appear on the application user's screen. While the application could issue a CLI command to take down the menu so that it would appear only briefly, another mechanism exists that will prevent it from appearing at all.

To prevent the main AVS menu from appearing, you modify the application to include two additional lines. For example, *app1* would appear as:

```
main()
{
   int argc;
   char *argv[3];
   extern char *strdup();
   extern int AVSno_main_menu;    <----
                                          new lines
   AVSno_main_menu = 1;           <----
   argc = 3;
   argv[0] = strdup("avs");
   argv[1] = strdup("-cli");
   argv[2] = strdup("script -play app1.scr");
   avs(argc, argv);
}
```

The **AVSno_main_menu** symbol is defined in *libavs.a*.

Licensed Applications: Defining a FLEXlm Licensing FEATURE String

By default, an application linked to *libavs.a* has the FLEXlm licensing FEATURE name "AVS-Runtime".

You should give your application its own unique FEATURE name:

- before you deploy the application, OR
- in order to enable the Network Editor in RuntimeB versions of Developers AVS.

The limits on the feature name are:

- No length limit, within reason.
- Any ASCII character except blanks and various kinds of quote marks.
- Case *is* significant.
- If you have the RuntimeB Developers AVS product that enables the Network Editor, then the FEATURE string *must* contain the suffix "-NE".

To define the FEATURE name, you declare the external character string variable **avs_runtime_feature**, and assign it a string. The **avs_runtime_feature** variable is defined in *libavs.a*.

For example:

```
extern char *avs_runtime_feature;         <----New line

main()
{
   int argc;
   char *argv[3];
   extern char *strdup();
   extern int AVSno_main_menu;

   AVSno_main_menu = 1;
   avs_runtime_feature = "AVS-Myapp";     <---New line
   argc = 3;
   argv[0] = strdup("avs");
   argv[1] = strdup("-cli");
   argv[2] = strdup("script -play app1.scr");
   avs(argc, argv);
}
```

Once the application contains a FEATURE line assignment other than the default "AVS-Runtime", then you must have a *license.dat* file that contains that FEATURE line and a valid license code from Advanced Visual Systems before you will be able to run the application.

Unlicensed Applications (OEM-Supplied Licensing)

If you have negotiated a special agreement with Advanced Visual Systems, then you may have the version of Developers AVS that permits you to install your own licensing mechanism. This version does not contain FLEXlm licensing.

Hook for License Validation

The following sample program shows how you would integrate your own licensing mechanism into an AVS application. **avs_periodic_check_func** is an AVS-provided function pointer (*libavs.a*) that lets you periodically check to see if a license is available. Initially set to NULL, if you reassign this to your own licensing-checking mechanism, AVS will call it every second.

Note that, although your license-checking mechanism is being called every second, you do not have to perform a complete license validation at every call. For example, AVS's own internal licensing mechanism is called frequently. However, it keeps track of a system clock and only performs a complete validation after a much longer interval has elapsed.extern int (*avs_periodic_check_func)();   <---the AVS-supplied function

```
static                              <---your own licensing function
check()
{
   printf("periodically checking...\n");
}

main()

{
  int argc;
  char *argv[3];
  extern char *strdup();

  argc = 3;
  argv[0] = strdup("avs");
  argv[1] = strdup("-cli");
  argv[2] = strdup("script -play test.scr");

  avs_periodic_check_func= check;         <---set AVS called function to your
                                  function
  avs(argc, argv);
}
```

Unlicensed AVS: Enabling the Network Editor

If you have purchased both:

- The RuntimeB (with Network Editor) version of Developers AVS, and
- An unlicensed version of Developers AVS,

then you can enable the Network Editor in your applications using the **avs_network_editor_enabled** variable as shown in the example below. (In RuntimeB, the Network Editor is still disabled by default.) This symbol is defined in *libavs.a*.

```
extern int avs_network_editor_enabled;

main()

{
   int argc;
   char *argv[3];
   extern char *strdup();

   argc = 3;
   argv[0] = strdup("avs");
   argv[1] = strdup("-cli");
   argv[2] = strdup("script -play test.scr");

   avs_network_editor_enabled= 1;

   avs(argc, argv);
}
```

Application Makefiles

The file *examples/Makefile* is a good example of how your makefile should look. It defines a number of macros that in turn define what include directories should be used and what libraries are needed to link an application. It is useful to examine which parts of the Makefile are important when making your own Makefile.

```
# This include file is found in the standard AVS release and contains
# platform specific flags and options needed in compiling ordinary
# AVS modules and programs.

# If AVS is not installed in /usr/avs, then set the AVS_PATH environment
# variable and override it during the make as in:
#    setenv AVS_PATH /my_dir/avs
#    make -e

AVS_PATH=$(ROOT)/usr/avs
INC_FILE=$(AVS_PATH)/include/Makeinclude
include $(INC_FILE)

AVS_LIBS = $(AVS_PATH)/lib

# List of buildable examples:
#   EX_APPS may be reduced by some platforms which don’t support Motif or X11R4

EX_APPS= app1 app2 app2_mod app3 app3_mod app4 app4_mod app5 app5_mod \
         app6 app6_mod

# Port includes are an optional list of directories in which to find
# Motif and X11 include files if they are not in the standard locations
```

```
PORT_INC=

# Developer definitions are located in a platform specific include file
# overriding libraries or include directories

DEV_DIR=$(AVS_PATH)/developer
include $(DEV_DIR)/include/Makelib.$(PORT)

######################################################################
# Generic library definitions and compiler flags

AVS_INC = -I. -I$(AVS_PATH)/include
CFLAGS=$(G) -D_MEM_DEFS_defined $(AVS_INC) $(PORT_INC)

DEV_LIBS=-L$(DEV_DIR)/lib -lavs -lobj -lren -lapp -lpic \
-lrf -lren -lflow_c -llui -lgeom -lutil -lspaceball

######################################################################
# Module libraries: needed for AVS modules (similar to /usr/avs/examples)
BASELIBS=-lgeom -lutil -lm
FLOWLIBS=-L$(DEV_DIR)/lib -lflow_c $(BASELIBS) $(LASTLIBS)
CSIMLIBS=-L$(DEV_DIR)/lib -lsim_c $(BASELIBS) $(LASTLIBS)

######################################################################
# Example applications

all: $(EX_APPS)

app1:   app1.o
        cc $(CFLAGS) -o app1 app1.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)

app2:   app2.o
        cc $(CFLAGS) -o app2 app2.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)

app2_mod: app2_mod.c
        $(CC) $(CFLAGS) -o app2_mod app2_mod.c $(CSIMLIBS)

app3_mod: app3_mod.o app3_menu.o app3_popups.o app3_callb.o
        $(CC) $(CFLAGS) -o app3_mod app3_mod.o app3_menu.o app3_popups.o \
        app3_callb.o $(MLIBS) $(CSIMLIBS)

app3:   app3.o
        cc $(CFLAGS) -o app3 app3.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)
app4_mod: app4_mod.o app4_menu.o app4_callb.o
        $(CC) $(CFLAGS) -o app4_mod app4_mod.o app4_menu.o \
        app4_callb.o $(MLIBS) $(CSIMLIBS)

app4:   app4.o
        cc $(CFLAGS) -o app4 app4.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)

app5_mod: app5_mod.o app5_menu.o app5_callb.o
        $(CC) $(CFLAGS) -o app5_mod app5_mod.o app5_menu.o \
        app5_callb.o $(MLIBS) $(CSIMLIBS)

app5:   app5.o
```

```
        cc $(CFLAGS) -o app5 app5.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)

app6_mod: app6_mod.o app6_menu.o app6_callb.o
        $(CC) $(CFLAGS) -o app6_mod app6_mod.o app6_menu.o \
        app6_callb.o $(MLIBS) $(CSIMLIBS)

app6:   app6.o
        cc $(CFLAGS) -o app6 app6.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)

.c.o:
        cc $(CFLAGS) $).c -c

clean:
        rm -f $(EX_APPS) *.o
```

The initial Makefile macros are similar to those found in *$AVS_PATH/examples/Makefile* and should be used in most AVS related Makefiles. INC_FILE, AVS_LIBS and AVS_INC all provide a way to reference the standard AVS product area whether it is in */usr/avs* or some other location defined by *$AVS_PATH*. The *Makeinclude* file is included to pick up platform-specific definitions provided to all AVS users. It defines PORT which selects which developer *Makelib* file is required.

Some of the Makefile features are specific to the examples provided in this release, in particular the EX_APPS macro which allows particular platforms to avoid building *app3, app4, app5*, and *app6* if they do not suport Motif.

As mentioned above, the DEV_DIR macro provides a way to specify where the developer product is installed. This is important since any developer application must link with the *libavs.a* library, any other libraries (*liblui*, *libgeom*, etc. in the *avs_developer/lib* directory, and the platform specific libraries defined in the *Makelib.$(PORT)* file found in the *avs_developer/include* directory. The Makelib file defines two sets of libraries—GEOMLIBS and XLIBS—which an application needs in addition to DEV_LIBS, which is defined above.

To develop your own application, it would be easiest to make a copy of this Makefile and insert your own application make lines in place of *app1* and then customize the makefile as required. This will provide the greatest portability when moving your application to another platform.

Deploying an Application

Once the application is built and ready to deliver, the developer combines the application and a subset of the standard AVS product into a deliverable product. The Developers AVS product permits the developer to copy a select portion of the standard AVS release and distribute it along with the application program. The recommended approach is to create a product "installation image" into which required AVS files are copied along with the Developers application. This installation image is then copied onto a CD in order to deliver the product. How the product CD is made may vary from one platform to another and one developer to another. The *tar* archive utility is the most common facility used for this purpose.

The Developers AVS product provides some assistance in building up the staging area. In the *tools* directory there is shell script, *install_files*, that will transport files from the */usr/avs* or *$AVS_PATH* area to the staging area, making sure that the same relative pathnames are preserved. This script can also be used to transport developer files to the same area. To use it, go to the *tools* directory and find the *install_files* shell script. Its usage message provides basic instructions:

```
install_files [-S src_dir] [-D dest_dir] [-F files] [-R]

install_files copies all of the necessary files from the source area to an
installed directory. It will create any necessary intermediate directories,
set file group and ownership to root unless otherwise specified,
and set file protection to 644 for non-executable files, 755 for executable
files. It uses a transport list file specifying the relative source
pathname of files to be transported. It can be used to copying out
selected sections of the AVS distribution area or to copy file from
a user's development area.



Flags:
   -S dir          | specify the directory that the source resides in.
                   | By default this is /usr/avs.
   -D dir          | specify the destination directory.  This is
                   | the prefix to the partial filename. There is no default.
   -F files        | specify the file containing the transport file list
                   | By default, this is STD_AVS_FILES.
                   | You can subset the file STD_AVS_FILES
                   | or provide a list of your own non-AVS files.
   -R              | Ordinarily files are copied as root group and owner
                   | This flag will turn this off.

Example:

   install_files -S /mypath/usr/avs -D /usr/mydev_area -R
```

The *STD_AVS_FILES* file is a list of files that may be transported from the */usr/avs* or *$AVS_PATH* area to the developers product CD. Files that are not on this list *may not* be transported without express written permission from Advanced Visual Systems Inc. The developer may wish to edit down a copy of this list to reduce how much is delivered; the reduced list can be specified using the **-F** option.

The developer can also set up a transport list file to transport the application program and any other required files. Each line in the file is a partial pathname of the file being transported, for example, "app1". During transport the full pathname is made up as $(SOURCE)/filename and transported to $(DEST)/filename.

The *avs_developer/tools* directory has sample transport lists for each of the example applications. Installing the *app2* example into a staging area is a good example of how to use the *install_files* script. Instead of using the full *STD_AVS_FILES* list, only the *AVS_FILES.ex* subset is used since these are the only files required. To get *app2* set up in */tmp* as a standalone application, type the following:

```
mkdir /tmp/app2_dir
install_files -S ../examples -F app2_files -D /tmp/app2_dir -R
install_files -S /usr/avs -F AVS_FILES.ex -D /tmp/app2_dir -R
```

This has installed both the *app2* application and its required files, as well as the subset of the standard AVS release that it requires. The -*S* option specified where the source files were coming from, the -*F* option specified the transport list, and -*D* gave the destination directory. The -*R* option is used to override the need to be the superuser. To verify that *app2* is truly standalone, do the following:

```
mv $AVS_PATH $AVS_PATH.save
```

and then try the application, first relying on *$AVS_PATH* being present and then providing the **-app_path** option to *app2* to get it to set its paths properly. Make sure to set *$AVS_PATH.save* back after you are done.

```
/tmp/app2_dir/app2

/tmp/app2_dir/app2 -app_path /tmp/app2_dir
```

Optimizing Geometry Modules: Linking Modules with the Kernel

AVS data flow is diagrammed in Figure 3-1. (For a complete description of this diagram, see p. 1-8 of the *AVS Developer's Guide*.)

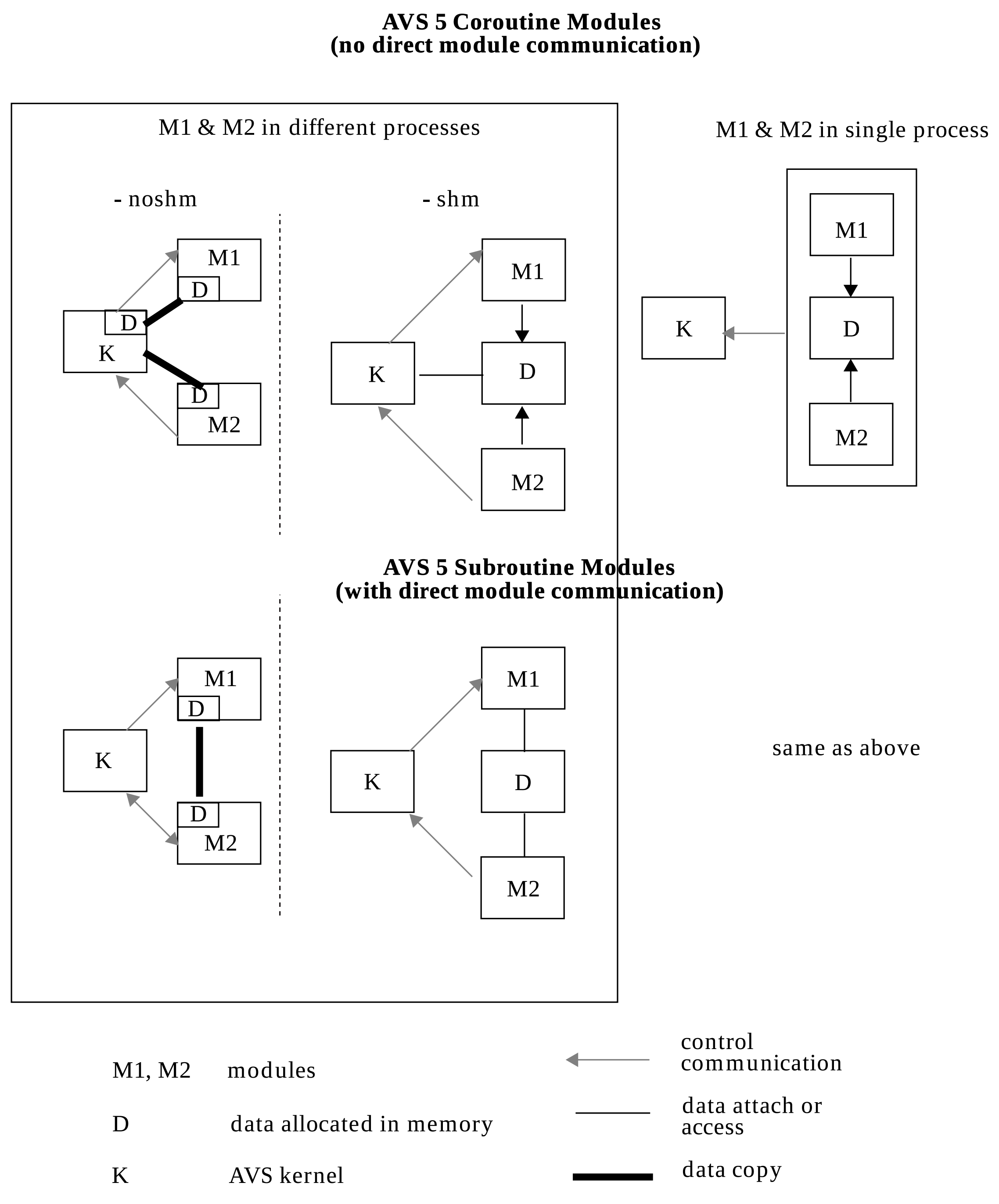

Figure 3-1 Data Flow Between Kernel and Modules

AVS uses shared memory to pass field and UCD data between modules in different processes (center column of figure). It does not use shared memory for geometry and chemistry data. Geometry and chemistry data is passed between modules in different processes by copying the data through a Unix socket (left column of figure).

Thirteen AVS modules are "builtin" to the AVS kernel. These modules are: **generate colormap**, **display image**, **geometry camera, geometry viewer**, **graph viewer**, **image viewer** (supported); and **colormap manager**, **image manager**, **volume manager**, **display pixmap**, **render geometry**, **render manager**, **transform pixmap** (unsupported). Note that these modules include the various viewers, notably the Geometry Viewer.

The data flow / memory utilization situation with geometry data that is being sent to the Geometry Viewer for display is the case diagrammed in the lower left corner of the figure, where M1 is the module producing the geometry and M2 is effectively inside K, the AVS kernel.

This data flow configuration cannot be altered with the primary AVS product. However, it can be altered with the Developers AVS product. In Developers AVS, the AVS kernel exists as the *lib/libavs.a* library. It is possible to link a mapper module that creates geometry data in with the AVS kernel producing a single process. This creates the data flow / memory utilization situation similar to that portrayed in the right column of the chart: M1 is the module producing the geometry, and M2 has become K, the AVS kernel. One copy of the data exists rather than two, and there is no time delay caused by copying the data through the Unix socket.

The actual, practical performance improvement realized by this procedure will vary from system to system. In general, it is only useful for very large geometries. For example, in tests on a system with 32 megabytes of main memory, the improvement became subjectively noticeable when the geometry contained in excess of 30,000 triangles.

You should experiment to see if the mechanism is worthwhile for the datasets and platforms used with your application.

Procedure Summary

The basic procedure is to link the geometry-producing module in with the application’s main routine, rather than by itself or with the suite of modules that make up the rest of your application’s network. Since the application’s main routine is linked with the AVS kernel, the geometry-producing module is also linked with the AVS kernel. All execute as one process.

Next, have the application’s main routine initialize the module with **AVSmodule_from_desc**. This differs from the usual procedure in which the kernel initializes the module by calling the initialization function in the module.

Some other file housekeeping modifications may also need to be made.Each step is described in greater detail in the sections below.

Changes to the Module's Source Code

Comment out, delete, or "ifdef" the module's initialization function.

For example, the **ucd to geom** module's description and initialization functions occur at the end of the module. They would be modified as follows:

```
/*-----------------------------------------------------*
 *          ****  ucd_to_geom_desc  ****               *
 *-----------------------------------------------------*/

ucd_to_geom_desc()
{

   int ucd_to_geom(), param;


   AVSset_module_name ("ucd to geom", MODULE_MAPPER);

   AVScreate_input_port ("Input", "ucd", REQUIRED);
   AVScreate_input_port ("Input Contour", "field 1D 3-vector real", OPTIONAL);
   AVScreate_input_port ("Cell Colors", "field 1D 3-vector real", OPTIONAL);
   AVScreate_output_port ("Output", "geom");
               .
               .
               .
/*#ifndef sep_exe                                      */
/*   AVSset_module_flags (COOPERATIVE);                */
/*#endif                                               */
}

/*-----------------------------------------------------*
 *         ****  initialization function  ****         *
 *-----------------------------------------------------*/


#ifdef sep_exe                               <---initialization made conditional
AVSinit_modules()                                on a "separate executable"
{                                                preprocessor directive
   int ucd_to_geom_desc();

   AVSmodule_from_desc(ucd_to_geom_desc);
}
#endif                                       <---close conditional
```

In truth, this step is not strictly necessary since the kernel will never actually call the module's initialization function if the application's main is correct. However, it is better to make the difference explicit than to rely upon a coincidence in the kernel code.

A module without an initialization function cannot be linked against the AVS libraries by itself. For this reason, you may prefer to surround the initialization function with an **ifdef** as shown in the example, rather than comment it out or delete it. This will allow you to compile and link the module by itself for testing purposes, as well as bundled in with the kernel.

Note that the **AVSset_module_flags(COOPERATIVE)** call has been commented out. A module that is linked with the kernel must be *coded* to be COOPERATIVE. If there will be more than one instance of the module in existance during an AVS session, then it must also be *coded* to be REENTRANT. (The meaning of these terms is discussed in the *AVS Developer's Guide* in the "Advanced Topics" chapter under "Multiple Modules in a Single Process.") It does not matter which, if any, parameters are given to an **AVSset_module_flags** routine because the portion of the AVS kernel that pays attention to these flags is the part that starts external processes.

No other changes are necessary to the module's source code.

Changes to the Application Main Routine

Modify the application's main routine that calls AVS to initialize the module by calling its description function. This is the key to the procedure. For example, if *app1.c* were attempting to include *ucd_to_geom.c*, it would read as follows:

```
main()
{
   int argc;
   char *argv[3];
   extern char *strdup();
   int ucd_to_geom_desc();               <-----the module's description function

   AVSmodule_from_desc(ucd_to_geom_desc);  <--initialize the module
   argc = 3;
   argv[0] = strdup("avs");
   argv[1] = strdup("-cli");
   argv[2] = strdup("script -play myapp.scr");
   avs(argc, argv);
}
```

Changes to the Module Library File

If the geometry-producing module is part of a module library file, its description should be changed from "external" to "builtin" and the remaining description information should be deleted.

For example, the line describing **ucd to geom**:

```
external "ucd to geom" 2 "ucd_multm" 3 2 "Input" 1 "ucd" "Input Contour" 0 \
        "field 1D 3-vector real" 1 "Output" 0 "geom" 7 "Shrink" "boolean" \
        "Shrink Factor" "integer" "Geometry Display Mode" "string" "mode" \
        "choice" "Explode Materials" "boolean" "Explode Factor" "integer" \
        "Save Geometry" "boolean"
```

becomes just:

```
builtin "ucd to geom"
```

Changes to the Network File

The CLI **module** command in the application's network file that loads the module may have **-ex *filename*** clauses directing AVS to load the module's binary from a file. The **-ex** clauses should be deleted. The rest of the line should remain.

Changes to Makefiles

The line(s) in the module's Makefile that define how to make the geometry-producing module should be changed so that the module is compiled only, not linked. (Without an initialization function, the module will not link against *libflow_c.a.*)

The application's Makefile should be changed to link the module's object file in with the application. For example:

```
app1:  app1.o
       cc $(CFLAGS) -o app1 app1.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)
```

becomes:

```
app1:  app1.o
       cc $(CFLAGS) -o app1 app1.o ucd_to_geom.o $(DEV_LIBS) $(GEOMLIBS) $(XLIBS)
```

Special Case: Duplicate Module Names

If you give one of your modules the same name as an existing module (as specified by the **AVSset_module_name** routine), ambiguities can result as to which version of the module, the builtin or the external, is actually initialized during the course of an application session. (See the "Documentation Clarifications/Corrections" chapter in the *AVS 5 Update* manual, in the section "Loading Modules with the CLI module Command: the -ex Option" for an explanation of how AVS chooses which module to use.

The simple solution is to be sure your modules have unique names. If you must use a duplicate name, then:

1. Create a separate module library to hold your module(s). Its module description should be "builtin" as described in the previous "Module Library File" section.
2. If you never want the existing module invoked, edit its module library file to read "builtin" as described.
3. Load the new module library file as part of the arguments sent to AVS by the application's main routine:

```
main()
{
   int argc;
   char *argv[5];
   extern char *strdup();
   int MODarbitrary_slicer();
   int MODprobe();

   AVSmodule_from_desc(MODarbitrary_slicer);
   AVSmodule_from_desc(MODprobe);
   argc=5;
   argv[0] = strdup("avs");
   argv[1] = strdup("-library");
   argv[2] = strdup("upstr_lib");
   argv[3] = strdup("-cli");
   argv[4] = strdup("script -play upstream.scr");
   avs(argc,argv);
}
```

4. Make sure that your module library is the current module library when your module is initialized.

MODULE SOURCE

Developers AVS includes source to most non-builtin AVS modules. Documentation is provided through online files in the *module_source* directory.

Two Module Source Products

The AVS 5.5 Developers product contains two versions of the module source product AVS_MOD_SRC8 and AVS_MOD_SRC9. The AVS_MOD_SRC9 product is the version of files that were used to build the AVS modules for the 64 bit platforms (Compaq Tru64 UNIX and SGI 64 bit). The MOD_SRC product contains module source for all of the other platforms.

Compiling Modules

The file *module_source/README.mod_src* in the MOD_SRC installation directory explains how to make the AVS modules, both individually and as a single binary.

Module to Module Source Mapping

The file *module_source/mod_map* contains a table that shows which modules are in which module source files. For example, it shows that the source to the **compute gradient** module is actually in *modules/filters/generic/vex_filters.c*.

Module to Module Binary Mapping

The file *module_source/mod_map2* contains a table that shows which modules are in which module binary files. For example, it shows that the module **draw grid** is compiled and linked into the module binary file *sv_multm*.